

Computer Security Foundations Symposium 2011 (CSF '11)

Obstruction-free Authorization Enforcement

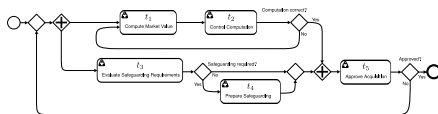
Aligning Security and Business Objectives



Motivation: Aligning security and business objectives

Business

- Modeled by workflows



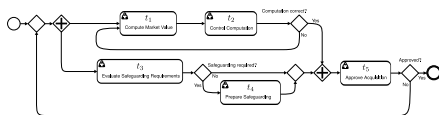
- Goal: “get the job done”

Doesn't matter who is executing a task – as long as it gets executed.

Motivation: Aligning security and business objectives

Business

- Modeled by workflows



- Goal: “get the job done”

Doesn't matter who is executing a task – as long as it gets executed.

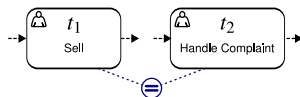
Security

- Authorizations constraining task executions

- Static authorizations
- Separation of Duty (SoD)



- Binding of Duty (BoD)



- Goal: no unauthorized task executions

Research objectives

- **Modeling & formalization**
 - Generic: not technology dependent
 - Applicable: can be translated to real systems
- **Challenge to link security & business world: novel notion of obstruction**
 - Limitation of business (control-flow) . . .
 - . . . caused by authorization policy (security),
 - generalizing deadlock

Research objectives

- **Modeling & formalization**
 - Generic: not technology dependent
 - Applicable: can be translated to real systems
- Challenge to **link security & business world**: novel notion of **obstruction**
 - Limitation of business (control-flow) . . .
 - . . . caused by authorization policy (security),
 - generalizing deadlock
- Goal: find **obstruction-free enforcement**
 - Decidability
 - Complexity
 - Approximation

Outline

1. Motivation

2. Release

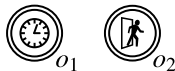
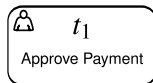
3. Obstruction

4. Related work

5. Conclusions

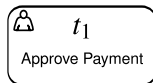
Basic objects

- \mathcal{U} is a set of **users**, e.g. $\mathcal{U} = \{\text{Alice, Bob, Claire}\}$
- \mathcal{T} is a set of **tasks**, e.g. $\mathcal{T} = \{t_1, t_2, \dots\}$
- $\mathcal{X} = \mathcal{T} \times \mathcal{U}$ is the set of **(task) execution events**.
 $(t, u) \in \mathcal{X}$ is written $t.u$ in CSP. e.g. $t_1.\text{Alice}$
- \mathcal{O} is a set of **points**, e.g. $\mathcal{O} = \{o_1, o_2, \dots\}$

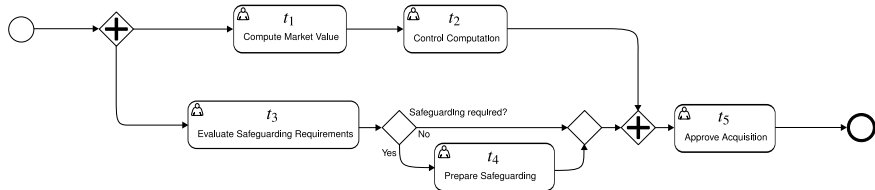


Basic objects

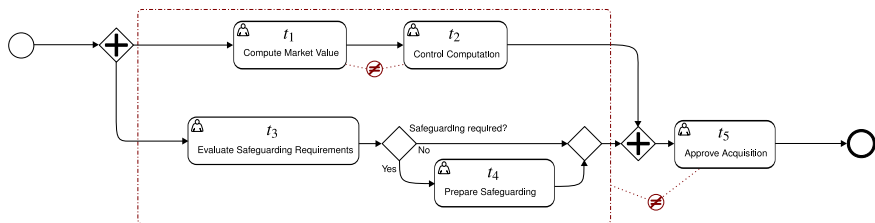
- \mathcal{U} is a set of **users**, e.g. $\mathcal{U} = \{\text{Alice, Bob, Claire}\}$
- \mathcal{T} is a set of **tasks**, e.g. $\mathcal{T} = \{t_1, t_2, \dots\}$
- $\mathcal{X} = \mathcal{T} \times \mathcal{U}$ is the set of **(task) execution events**.
 $(t, u) \in \mathcal{X}$ is written $t.u$ in CSP. e.g. $t_1.\text{Alice}$
- \mathcal{O} is a set of **points**, e.g. $\mathcal{O} = \{o_1, o_2, \dots\}$
- A trace $i \in (\mathcal{X} \cup \mathcal{O})^* \checkmark$ models a **workflow instance**.
e.g. $i = \langle t_1.\text{Alice}, o_1, t_2.\text{Bob}, t_1.\text{Claire}, \dots, \checkmark \rangle$



Scoping with release points

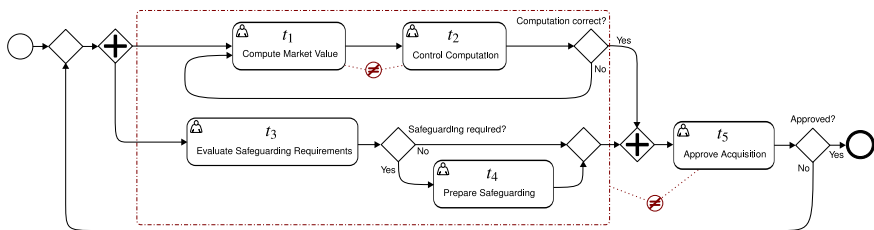


Scoping with release points



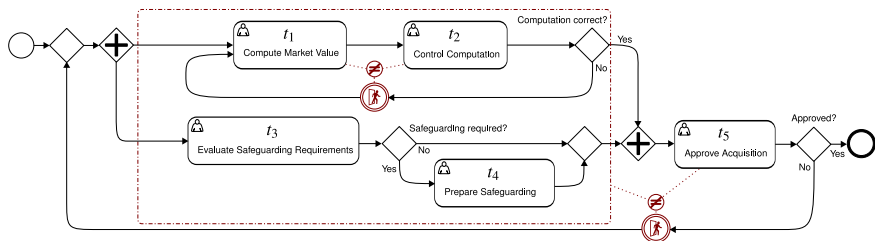
- Task executions relate users to task (instances).

Scoping with release points



- Task executions relate users to task (instances).
- Loops yield arbitrary number of instances per task

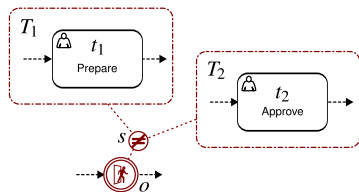
Scoping with release points



- Task **executions relate** users to task (instances).
- Loops yield **arbitrary number of instances** per task
- Release points **clear** task execution **history** w.r.t. constraint.
Thereby **release points scope constraints** to subsets of task instances.

Dynamic authorization constraints

SoD constraint $s = (T_1, T_2, O)$



SoD process $A_s(\mathcal{U}, \mathcal{U})$ for

$$A_s(U_{T_1}, U_{T_2}) = t : T_1.u : U_{T_1} \rightarrow A_s(U_{T_1}, U_{T_2} \setminus \{u\})$$

$$\square t : T_2.u : U_{T_2} \rightarrow A_s(U_{T_1} \setminus \{u\}, U_{T_2})$$

$$\square o : O \rightarrow A_s(\mathcal{U}, \mathcal{U})$$

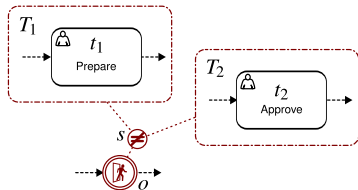
$$\square t : \mathcal{T} \setminus (T_1 \cup T_2).u : \mathcal{U} \rightarrow A_s(U_{T_1}, U_{T_2})$$

$$\square o : \mathcal{O} \setminus O \rightarrow A_s(U_{T_1}, U_{T_2})$$

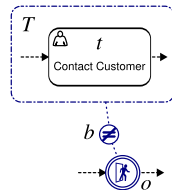
$$\square \text{SKIP}$$

Dynamic authorization constraints

SoD constraint $s = (T_1, T_2, O)$



BoD constraint $b = (T, O)$



SoD process $A_s(\mathcal{U}, \mathcal{U})$ for

$$A_s(U_{T_1}, U_{T_2}) = t : T_1.u : U_{T_1} \rightarrow A_s(U_{T_1}, U_{T_2} \setminus \{u\})$$

$$\square t : T_2.u : U_{T_2} \rightarrow A_s(U_{T_1} \setminus \{u\}, U_{T_2})$$

$$\square o : O \rightarrow A_s(\mathcal{U}, \mathcal{U})$$

$$\square t : \mathcal{T} \setminus (T_1 \cup T_2).u : \mathcal{U} \rightarrow A_s(U_{T_1}, U_{T_2})$$

$$\square o : \mathcal{O} \setminus O \rightarrow A_s(U_{T_1}, U_{T_2})$$

$$\square SKIP$$

BoD process $A_b(\mathcal{U})$ for

$$A_b(U) = \dots$$

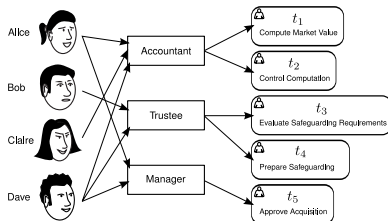
Static authorizations & authorization process

- **Static authorizations**
modeled as a relation $UT \subseteq \mathcal{U} \times \mathcal{I}$.
- **Static authorization process** A_{UT} for

$$A_{UT} = (t.u) : \{t'.u' \mid (u', t') \in UT\} \rightarrow A_{UT}$$

$$\square o : \mathcal{O} \rightarrow A_{UT}$$

$$\square SKIP$$



Static authorizations & authorization process

- Static authorizations

modeled as a relation $UT \subseteq \mathcal{U} \times \mathcal{I}$.

- Static authorization process A_{UT} for

$$A_{UT} = (t.u) : \{t'.u' \mid (u', t') \in UT\} \rightarrow A_{UT}$$

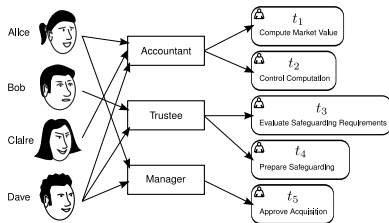
$$\square o : \mathcal{O} \rightarrow A_{UT}$$

$$\square SKIP$$

- Authorization policy $\phi = (UT, S, B)$

for a set of SoD constraints S and BoD constraints B

- Authorization process $A_\phi = A_{UT} \parallel \left(\parallel_{s \in S} A_s \right) \parallel \left(\parallel_{b \in B} A_b \right)$



Outline

1. Motivation

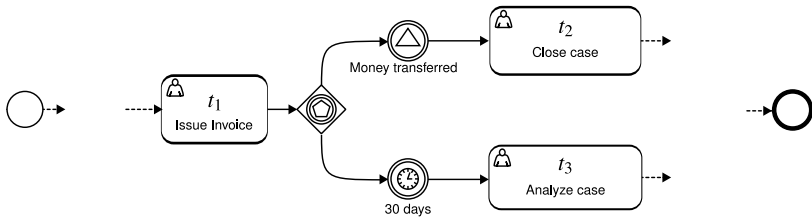
2. Release

3. Obstruction

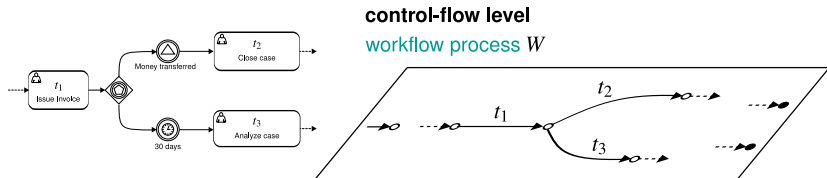
4. Related work

5. Conclusions

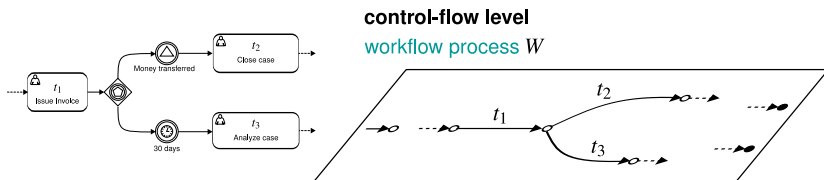
Obstruction



Obstruction



Obstruction



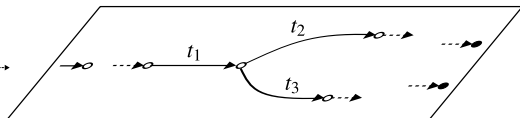
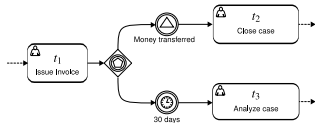
 Alice (a)

 Bob (b)


$$\pi = \{(t.u,t) \mid t \in \mathcal{T}, u \in \mathcal{U}\}$$

Obstruction

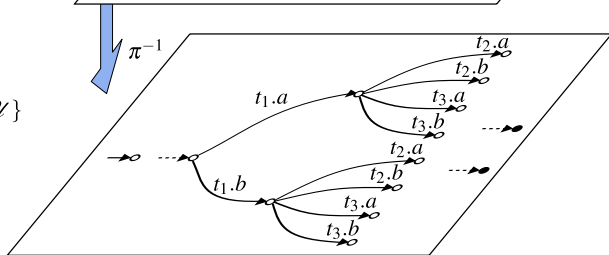
control-flow level
workflow process W



 Alice (a)

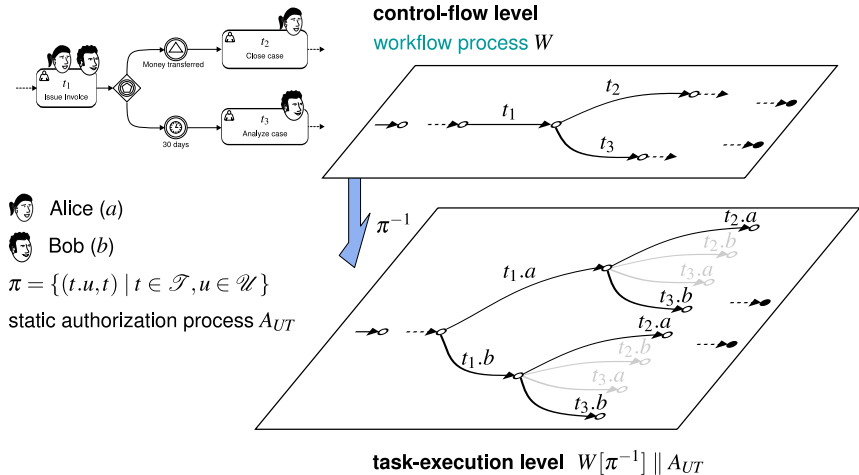
 Bob (b)

$$\pi = \{(t.u, t) \mid t \in \mathcal{T}, u \in \mathcal{U}\}$$

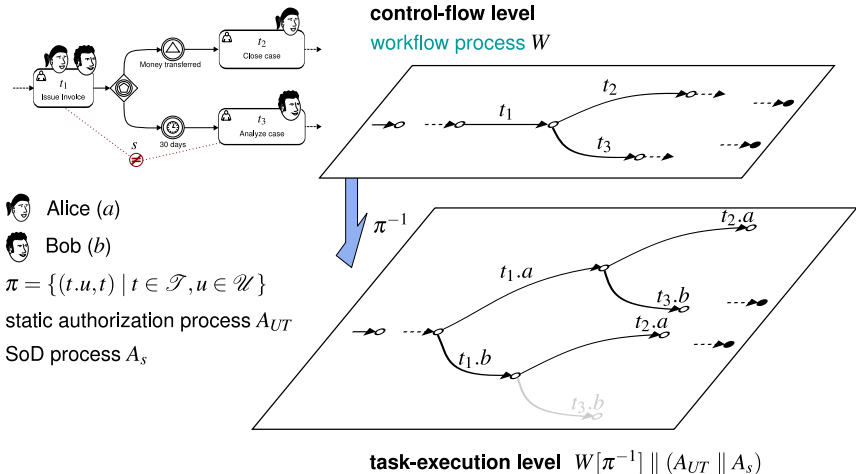


task-execution level $W[\pi^{-1}]$

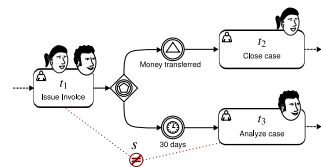
Obstruction



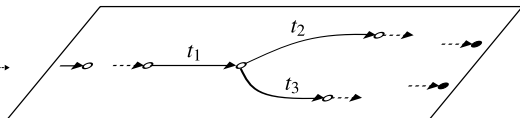
Obstruction



Obstruction



control-flow level
workflow process W



Alice (a)

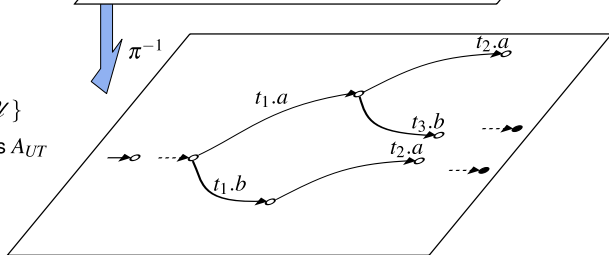
Bob (b)

$\pi = \{(t.u,t) \mid t \in \mathcal{T}, u \in \mathcal{U}\}$

static authorization process A_{UT}

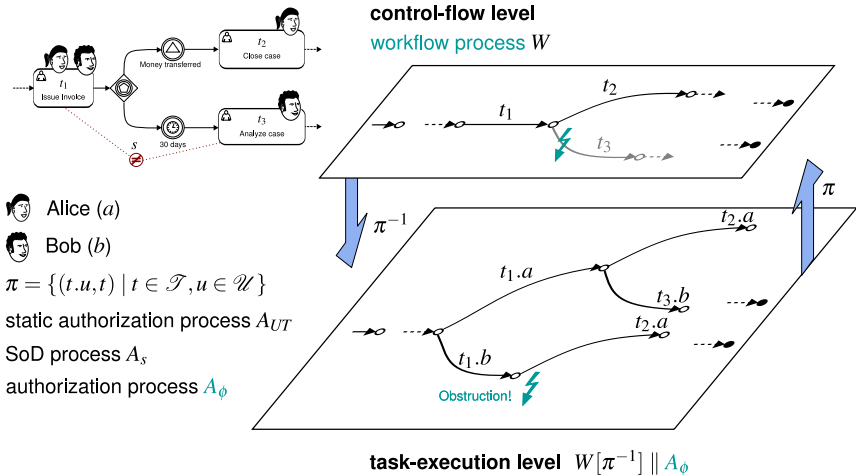
SoD process A_s

authorization process A_ϕ

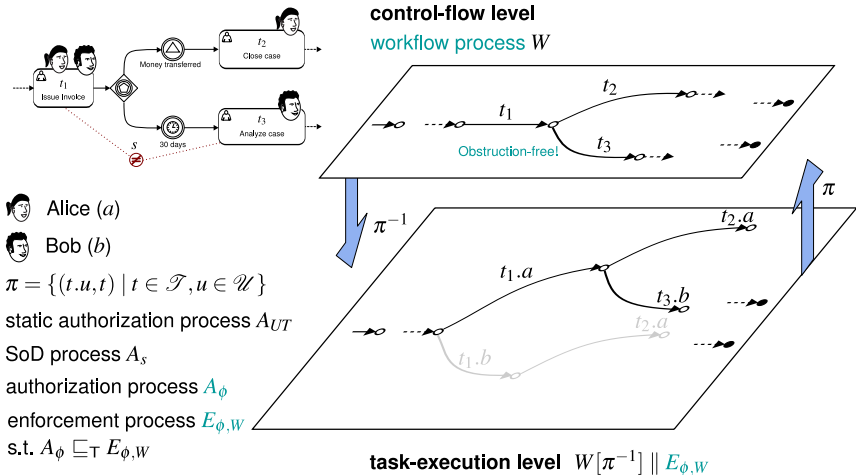


task-execution level $W[\pi^{-1}] \parallel A_\phi$

Obstruction



Obstruction



Enforcement process

Definition

For a workflow process W and an authorization policy ϕ .

An **enforcement process** for ϕ on W , written $E_{\phi,W}$,

is a (CSP) process that satisfies

- 1) $A_{\phi} \sqsubseteq_{\top} E_{\phi,W}$ and
- 2) $(W[\pi^{-1}] \parallel E_{\phi,W})[\pi] =_{\text{F}} W$.

Enforcement process

Definition

For a workflow process W and an authorization policy ϕ .

An **enforcement process** for ϕ on W , written $E_{\phi,W}$, is a (CSP) process that satisfies

- 1) $A_{\phi} \sqsubseteq_{\top} E_{\phi,W}$ and
- 2) $(W[\pi^{-1}] \parallel E_{\phi,W})[\pi] =_{\text{F}} W$.

Definition

Enforcement Process Existence Problem **EPE**

Given: A workflow process W and an authorization policy ϕ

Output: YES if there exists an enforcement process for ϕ on W , No otherwise.

EPE complexity

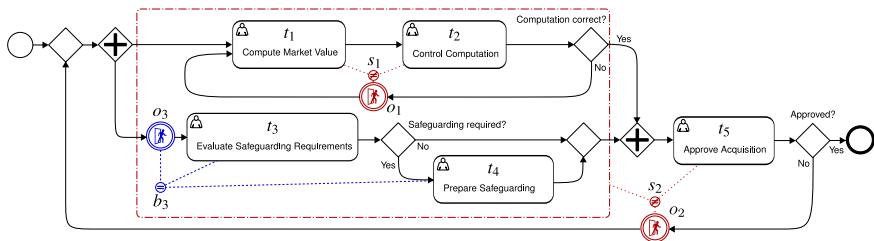
- **EPE** is **NP-hard**

Shown by reducing **k -COLORING** to **EPE**

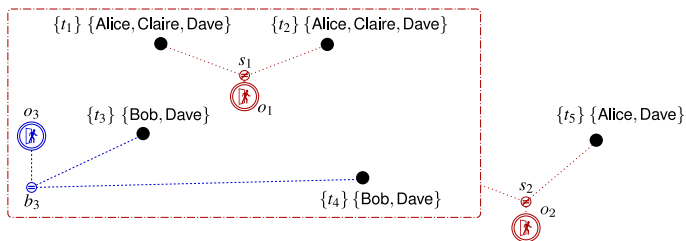
- **EPE** is **decidable** (if \mathcal{U} and W are finite)

- By enumerating the finitely many trace-refinements of A_ϕ
- Double exponential runtime complexity in the number of users and constraints

Approximation: EPE to LISTCOLORING

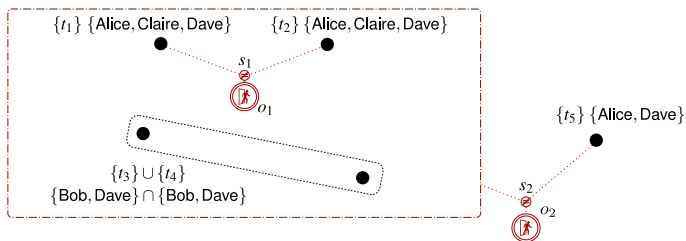


Approximation: EPE to LISTCOLORING



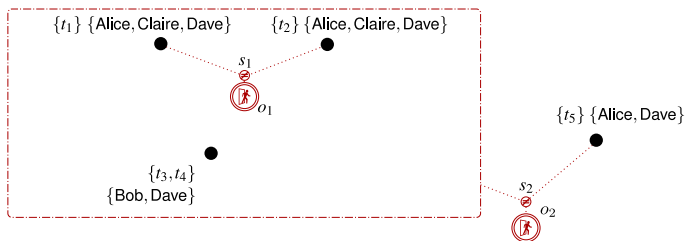
- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as node of a graph, annotated by set $\{u \mid (u, t) \in UT\}$

Approximation: EPE to LISTCOLORING



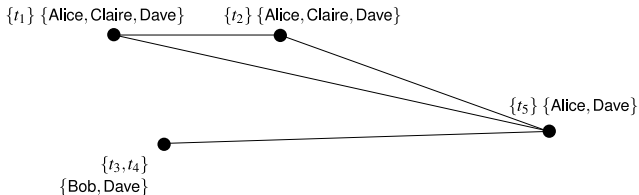
- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as **node** of a graph, annotated by set $\{u \mid (u, t) \in UT\}$
- Merge nodes linked by **BoD constraints** and intersect sets of users

Approximation: EPE to LISTCOLORING



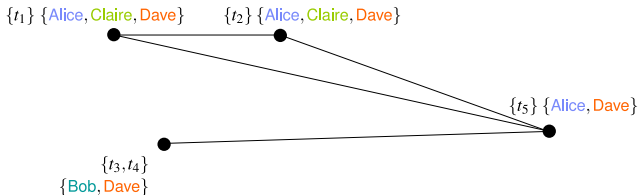
- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as **node** of a graph, annotated by set $\{u \mid (u, t) \in UT\}$
- **Merge** nodes linked by **BoD constraints** and intersect sets of users

Approximation: EPE to LISTCOLORING



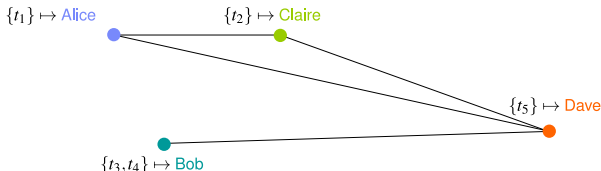
- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as **node** of a graph, annotated by set $\{u \mid (u, t) \in UT\}$
- **Merge** nodes linked by **BoD constraints** and intersect sets of users
- For every **SoD constraint**, add **edge** between repetitive tasks/nodes

Approximation: EPE to LISTCOLORING



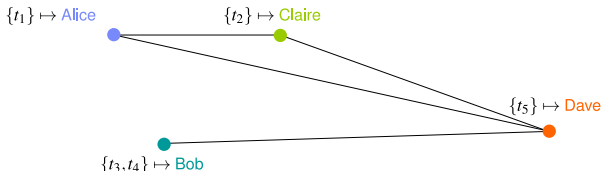
- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as **node** of a graph, annotated by set $\{u \mid (u, t) \in UT\}$
- **Merge** nodes linked by **BoD constraints** and intersect sets of users
- For every **SoD constraint**, add **edge** between repetitive tasks/nodes
- Consider sets of users as color “lists” of **LISTCOLORING** problem

Approximation: EPE to LISTCOLORING

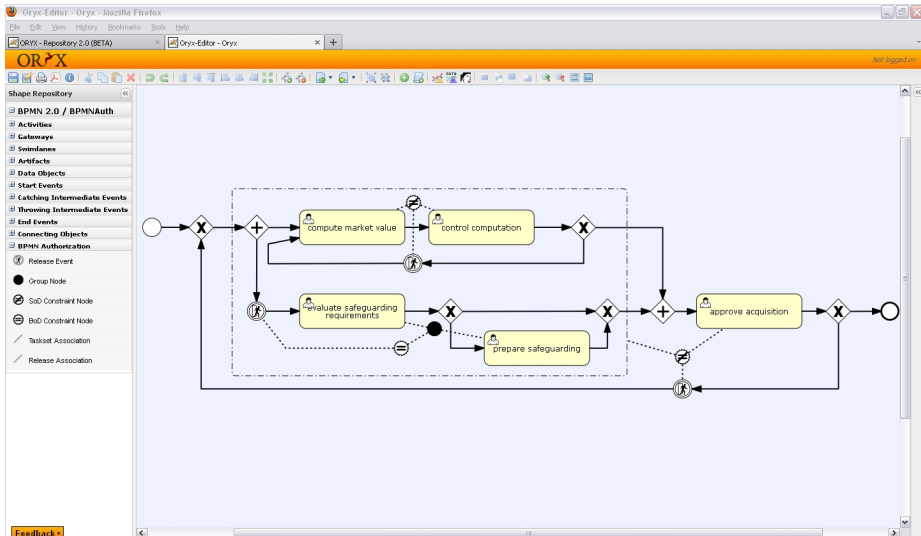


- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as node of a graph, annotated by set $\{u \mid (u, t) \in UT\}$
- Merge nodes linked by BoD constraints and intersect sets of users
- For every SoD constraint, add edge between repetitive tasks/nodes
- Consider sets of users as color “lists” of LISTCOLORING problem
- Solve LISTCOLORING instance

Approximation: EPE to LISTCOLORING



- For every task $t \in \mathcal{T}$, interpret $\{t\}$ as node of a graph, annotated by set $\{u \mid (u, t) \in UT\}$
- Merge nodes linked by BoD constraints and intersect sets of users
- For every SoD constraint, add edge between repetitive tasks/nodes
- Consider sets of users as color “lists” of LISTCOLORING problem
- Solve LISTCOLORING instance
- Coloring defines relation ex , s.t. $W[ex]$ is an enforcement process for ϕ on W .



Outline

1. Motivation

2. Release

3. Obstruction

4. Related work

5. Conclusions

Related work

- SoD in the context of RBAC. For us subsumed in UT.
- Well-formed constraints
 - **Satisfiability**: Given W, ϕ . Question: $\exists i. i^{\langle \checkmark \rangle} \in T(W[\pi^{-1}] \parallel A_\phi)$.^{1 2}
- Workflow expressivity
 - No loops or conditional execution.
E.g. list sequence of tasks¹ or partially order set of tasks²
- Constraints based on structures over users
 - E.g. roles¹ and relations between users²
- **Enforcement**: workflows provide more context than available to security automata³
- Business process community
 - **Soundness** of workflow models⁴

¹Bertino *et al.* *The Spec. and Enf. of Authorization Constraints in WfMS*. TISSEC 1999.

²Crampton. *A Reference Monitor for Workflow Sys. with Constrained Task Exec.* SACMAT '05.

³Schneider. *Enforceable Security Policies*. TISSEC 2000.

⁴Van der Aalst. *The Application of Petri Nets to Workflow Management*, 1998.

Outline

1. Motivation

2. Release

3. Obstruction

4. Related work

5. Conclusions

Conclusions

■ Contributions

■ Release

- Novel approach to **scope** authorization constraints
- Overcome previous limitation of **workflow expressivity**

■ Obstruction

- Generalization of **deadlock**
- Describing **miss-alignment** of security & business objectives
- **EPE**: Problem of finding **obstruction-free** authorization **enforcement**

- Both neither CSP nor BPMN-specific

Conclusions

■ Contributions

■ Release

- Novel approach to **scope** authorization constraints
- Overcome previous limitation of **workflow expressivity**

■ Obstruction

- Generalization of **deadlock**
- Describing **miss-alignment** of security & business objectives
- **EPE**: Problem of finding **obstruction-free** authorization **enforcement**

- Both neither CSP nor BPMN-specific

■ Applicability & outlook

- Implementing: Obstruction-freedom as “**spell-check**” in workflow modeling tool
- How to **tackle high complexity** of EPE
 - If input \mathcal{I} large: approximation typically successful
 - If input W large: consider decomposition
- Impact of **changing UT** on enforcement process